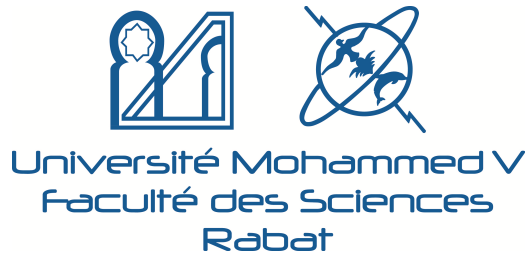


UNIVERSITÉ MOHAMMED V DE RABAT  
Faculté des sciences



Département d'informatique  
Intelligence artificielle et sciences numériques

---

PROJET D'OPTIMISATION  
COMBINATOIRE

---

Titre :

Problème de la coupe maximum

Présentée par :

MAKBOUL WIAM  
BOUTALAT HIBA

Livré le 13 février 2025.

Ms. SARA LAGHMITI Professeur à la Faculté des Sciences - Rabat *superviseur*

Année universitaire 2024-2025

# Introduction

Dans de nombreux domaines scientifiques et industriels, l'optimisation combinatoire joue un rôle essentiel en donnant des solutions efficaces à des problèmes complexes. Parmi eux, la coupe maximale en théorie des graphes est cruciale en raison de ses nombreuses applications.

Une coupe maximum est une coupe contenant au moins autant d'arêtes que n'importe quelle autre coupe. Il est commun que cette problématique se trouve dans des contextes tels que l'optimisation des réseaux, la segmentation d'images, l'intelligence artificielle et la bio-informatique. Cependant, en raison de sa classe de complexité (NP-difficile), il n'existe pas de solution précise pour tous les cas, ce qui inspire le développement de différentes approches heuristiques et métaheuristiques.

Ce rapport est prévu pour explorer bien le problème de la coupe maximum, en présentant ses fondements mathématiques, ses variantes, ainsi que les approches utilisées pour le résoudre. Nous analyserons également ses applications pratiques et les perspectives d'amélioration.

# Table des matières

|                                                                                 |           |
|---------------------------------------------------------------------------------|-----------|
| <b>Introduction</b>                                                             | <b>2</b>  |
| <b>Table des matières</b>                                                       | <b>3</b>  |
| <b>Table des figures</b>                                                        | <b>4</b>  |
| <b>1 Cadre théorique et modélisation</b>                                        | <b>1</b>  |
| 1.1 Formalisation mathématique et exemples . . . . .                            | 1         |
| 1.1.1 Définition mathématique du problème . . . . .                             | 1         |
| 1.1.2 Exemples de graphes et coupes . . . . .                                   | 1         |
| 1.2 Définition des variables et formulation du problème . . . . .               | 2         |
| 1.2.1 Variables de décision . . . . .                                           | 2         |
| 1.2.2 Fonction objectif . . . . .                                               | 2         |
| 1.2.3 Contraintes . . . . .                                                     | 3         |
| 1.3 Synthèse du cadre théorique . . . . .                                       | 3         |
| <b>2 Analyse des variantes du problème</b>                                      | <b>4</b>  |
| 2.1 Variantes selon la structure du graphe . . . . .                            | 4         |
| 2.1.1 Graphes non orientés vs. Graphes orientés . . . . .                       | 4         |
| 2.1.2 Graphes pondérés vs. non pondérés . . . . .                               | 4         |
| 2.1.3 Cas particuliers des graphes . . . . .                                    | 4         |
| 2.2 Variantes selon l'objectif de la coupe . . . . .                            | 6         |
| 2.3 Contraintes supplémentaires . . . . .                                       | 6         |
| 2.4 Synthèse . . . . .                                                          | 6         |
| <b>3 Applications et enjeux pratiques</b>                                       | <b>7</b>  |
| 3.1 Optimisation des réseaux et gestion des flux . . . . .                      | 7         |
| 3.1.1 Réseaux de télécommunications et informatiques . . . . .                  | 7         |
| 3.1.2 Transport et logistique . . . . .                                         | 8         |
| 3.2 Apports en Intelligence Artificielle et Apprentissage Automatique . . . . . | 8         |
| 3.2.1 Segmentation d'Images et Clustering . . . . .                             | 8         |
| 3.2.2 Applications en Bioinformatique . . . . .                                 | 8         |
| 3.3 Synthèse des applications . . . . .                                         | 10        |
| <b>4 Étude de cas et méthodologies de résolution</b>                            | <b>11</b> |
| 4.1 Présentation de l'étude de cas : Segmentation d'images par Max-Cut . . .    | 11        |
| 4.2 Exploration des approches exactes, heuristiques et métaheuristiques . . .   | 12        |
| 4.2.1 Approches Exactes . . . . .                                               | 12        |

|          |                                                                         |           |
|----------|-------------------------------------------------------------------------|-----------|
| 4.2.2    | Approches Heuristiques . . . . .                                        | 13        |
| 4.2.3    | Approches Métaheuristiques . . . . .                                    | 14        |
| 4.3      | Résolution numérique des approches de segmentation d'images par Max-Cut | 15        |
| 4.3.1    | Optimisation par Programmation Linéaire en Nombres Entiers (PLNE)       | 15        |
| 4.3.2    | Segmentation basée sur des Algorithmes Gloutons . . . . .               | 16        |
| 4.3.3    | Optimisation avec Algorithmes Génétiques . . . . .                      | 16        |
| 4.4      | Synthèse des Résultats . . . . .                                        | 16        |
| <b>5</b> | <b>Analyse critique et perspectives</b>                                 | <b>18</b> |
| 5.1      | Comparaison des méthodes . . . . .                                      | 18        |
| 5.2      | Conclusion et perspectives . . . . .                                    | 19        |
| <b>6</b> | <b>Implémentation Python</b>                                            | <b>20</b> |
| 6.1      | Bibliothèques Utilisées . . . . .                                       | 20        |
| 6.2      | Chargement de l'Image . . . . .                                         | 20        |
| 6.3      | Segmentation par Programmation Linéaire en Nombres Entiers (PLNE) .     | 20        |
| 6.4      | Segmentation par Algorithme Glouton . . . . .                           | 21        |
| 6.5      | Segmentation par Algorithme Génétique . . . . .                         | 22        |
|          | <b>Conclusion</b>                                                       | <b>24</b> |
|          | <b>Bibliographie</b>                                                    | <b>25</b> |

# Table des figures

|       |                                                                                                                                        |    |
|-------|----------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1.1 | Exemples de graphes . . . . .                                                                                                          | 2  |
| 2.1.1 | Exemple de graphe biparti . . . . .                                                                                                    | 5  |
| 2.1.2 | Exemple de graphe planaire . . . . .                                                                                                   | 5  |
| 3.1.1 | Représentation du reseau des 18 regions francaises graphe pour lequel les<br>noeuds . . . . .                                          | 7  |
| 3.1.2 | Le redéploiement technique et organisationnel des réseaux de messagerie<br>dans les territoires. Infrastructures de transport. . . . . | 8  |
| 3.2.1 | la segmentation d'images et le clustering dans le cadre du problème du<br>Max-Cut . . . . .                                            | 9  |
| 3.2.2 | Analyse des graphes en Bioinformatique . . . . .                                                                                       | 9  |
| 4.1.1 | Image d'entrée pour la segmentation . . . . .                                                                                          | 11 |
| 4.1.2 | Construction de graphe à partir de l'image . . . . .                                                                                   | 12 |
| 4.2.1 | Application de l'approche exactes pour Max-Cut . . . . .                                                                               | 13 |
| 4.2.2 | Application de l'approche heuristique pour Max-Cut . . . . .                                                                           | 14 |

|                                                                         |    |
|-------------------------------------------------------------------------|----|
| 4.2.3 Application de l'approche Métaheuristiques pour Max-Cut . . . . . | 15 |
|-------------------------------------------------------------------------|----|

# Chapitre 1

## Cadre théorique et modélisation

### 1.1 Formalisation mathématique et exemples

#### 1.1.1 Définition mathématique du problème

Le problème de coupe dans un graphe est un problème fondamental en théorie des graphes et en optimisation combinatoire. Il consiste à diviser un graphe en deux sous-ensembles disjoints tout en minimisant ou maximisant la somme des poids des arêtes coupées.

Soit un graphe non orienté  $\mathbf{G} = (V, E)$  avec :

- $V$  l'ensemble des sommets.
- $E$  l'ensemble des arêtes pondérées ou non pondérées.

Une **coupe** est définie par une bipartition de  $V$  en deux sous-ensembles  $S$  et  $T$  tels que :

$$S \cup T = V, \quad S \cap T = \emptyset$$

La **valeur de la coupe** est la somme des poids des arêtes ayant une extrémité dans chaque ensemble :

$$C(S, T) = \sum_{(u,v) \in E, u \in S, v \in T} w(u, v)$$

Où  $w(u, v)$  est le poids associé à l'arête  $(u, v)$ .

#### 1.1.2 Exemples de graphes et coupes

1. **Graphe non pondéré** : Si toutes les arêtes ont un poids égal à 1, la coupe revient à compter le nombre d'arêtes traversant la bipartition.

**Exemple** : Considérons un graphe avec quatre sommets  $V = \{A, B, C, D\}$  et les arêtes suivantes :

- $(A, B), (B, C), (C, D), (D, A), (A, C)$

Si l'on divise  $V$  en  $S = \{A, B\}$  et  $T = \{C, D\}$ , la coupe correspond aux arêtes  $(B, C)$  et  $(A, C)$ , donc  $C(S, T) = 2$ .

2. **Graphe pondéré** : Lorsque chaque arête a un poids différent, on cherche à maximiser ou minimiser la somme des poids des arêtes coupées.

**Exemple** : Supposons un graphe où les arêtes ont les poids suivants :

—  $(A, B) = 3, (B, C) = 5, (C, D) = 2, (D, A) = 4, (A, C) = 6$

Si  $S = \{A, B\}$  et  $T = \{C, D\}$ , la coupe est constituée des arêtes  $(B, C)$  et  $(A, C)$  avec une valeur totale de coupe  $C(S, T) = 5 + 6 = 11$ .

3. **Graphe orienté** : Les arêtes ont une direction, ce qui peut affecter les stratégies de coupe.

**Exemple** : Pour un graphe dirigé avec les arêtes  $(A \rightarrow B), (B \rightarrow C), (C \rightarrow D), (D \rightarrow A), (A \rightarrow C)$ , si l'on choisit la coupe  $S = \{A, B\}$  et  $T = \{C, D\}$ , seules les arêtes dirigées de  $S$  vers  $T$  sont comptabilisées, soit  $(B \rightarrow C)$  et  $(A \rightarrow C)$ .

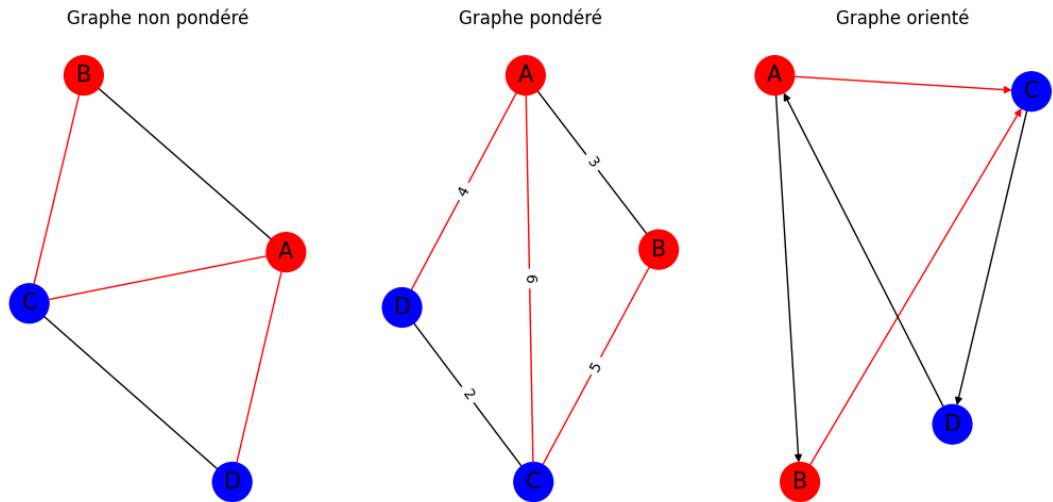


FIGURE 1.1.1 – Exemples de graphes

## 1.2 Définition des variables et formulation du problème

### 1.2.1 Variables de décision

Pour formaliser le problème, on définit une variable binaire associée à chaque sommet :

$$x_i = \begin{cases} 1, & \text{si le sommet } i \text{ appartient à } S \text{ (premier ensemble)} \\ 0, & \text{sinon} \end{cases}$$

Une arête  $(u, v)$  est coupée si et seulement si  $x_u \neq x_v$ .

### 1.2.2 Fonction objectif

Dans le cas de la **coupe maximale**, on cherche à maximiser :

$$\max \sum_{(u,v) \in E} w(u,v) |x_u - x_v|$$

Dans le cas de la **coupe minimale**, l'objectif devient :

$$\min \sum_{(u,v) \in E} w(u,v) |x_u - x_v|$$

### 1.2.3 Contraintes

- **Contraintes de partitionnement** : chaque sommet appartient à un seul des deux ensembles.
- **Conditions de connectivité** (dans certains cas) : on impose que les sous-ensembles restent connectés.

## 1.3 Synthèse du cadre théorique

Dans cette partie, nous avons établi une notion mathématique du problème du coupe et décrire comment modéliser ses différences instances à l'aide de variables décisionnelles et d'une fonction objectif. Cette modélisation facilite l'application d'algorithmes d'optimisation en vue de résoudre le problème de façon efficace. Dans les chapitres suivantes, nous présentons les différentes approches de résolution, ainsi que des exemples concrets d'application.

# Chapitre 2

## Analyse des variantes du problème

### 2.1 Variantes selon la structure du graphe

#### 2.1.1 Graphes non orientés vs. Graphes orientés

| Type de Graphe              | Caractéristiques                                    | Impact sur la coupe                                                                           |
|-----------------------------|-----------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <b>Graphes non orientés</b> | Les arêtes n'ont pas de direction.                  | Une coupe compte <b>toutes les arêtes</b> reliant deux sous-ensembles.                        |
| <b>Graphes orientés</b>     | Les arêtes ont une direction ( $u \rightarrow v$ ). | Seules les <b>arêtes dirigées</b> d'un ensemble à un autre sont comptabilisées dans la coupe. |

TABLE 2.1 – Comparaison entre graphes non orientés et graphes orientés

#### 2.1.2 Graphes pondérés vs. non pondérés

| Type de Graphe              | Caractéristiques                                                 | Impact sur la coupe                                                                  |
|-----------------------------|------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <b>Graphes pondérés</b>     | Chaque arête a un <b>poids</b> (coût, capacité, distance, etc.). | La <b>valeur de la coupe</b> dépend de la <b>somme des poids</b> des arêtes coupées. |
| <b>Graphes non pondérés</b> | Les arêtes n'ont <b>pas de poids</b> .                           | Seul le <b>nombre d'arêtes coupées</b> est pris en compte.                           |

TABLE 2.2 – Comparaison entre graphes pondérés et non pondérés

#### 2.1.3 Cas particuliers des graphes

Quelques types des graphes ont des caractéristiques spécifiques qui peuvent affecter l'approche adoptée pour résoudre les problèmes de coupes. Ces graphes ont souvent des structures qui simplifient ou compliquent certains calculs ou algorithmes. Voici quelques exemples :

- **Graphes bipartis** : Un graphe est dit biparti si son ensemble de sommets peut être divisé en deux sous-ensembles disjoints  $U$  et  $V$  tels que chaque arête ait une extrémité dans  $U$  et l'autre dans  $V$ . Un graphe biparti permet notamment de représenter une relation binaire.

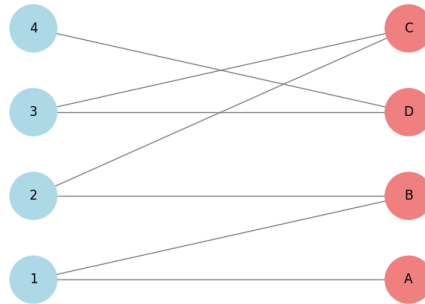


FIGURE 2.1.1 – Exemple de graphe biparti

#### Impact sur les problèmes de coupe

Dans les graphes bipartis, les problèmes de coupe deviennent souvent plus simples à résoudre, car les arêtes à couper n'existent qu'entre les deux sous-ensembles. Cela réduit les possibilités et facilite l'optimisation de la coupe, notamment pour des objectifs comme la coupe minimale.

- **Graphes planaires** un graphe planaire est un graphe qui a la particularité de pouvoir se représenter sur un plan sans qu'aucune arête (ou arc pour un graphe orienté) n'en croise une autre. Autrement dit, ces graphes sont précisément ceux que l'on peut plonger dans le plan, ou encore les graphes dont le nombre de croisements est nul.

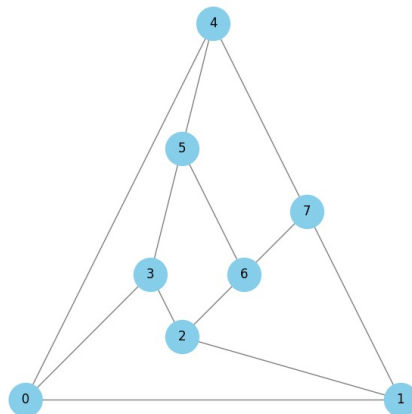


FIGURE 2.1.2 – Exemple de graphe planaire

#### Impact sur les problèmes de coupe

Les graphes planaires possèdent des propriétés structurelles particulières qui influencent les algorithmes de coupe. Grâce à leur organisation sans croisements d'arêtes, certaines opérations de partitionnement et de séparation peuvent être optimisées.

## 2.2 Variantes selon l'objectif de la coupe

- **Coupe minimum** : Le but est de minimiser la somme des poids des arêtes coupées. Cette approche est particulièrement utile en optimisation de réseaux.
- **Coupe maximum** : Le but est de maximiser la somme des poids des arêtes coupées. Cette approche est souvent utilisée en clustering et en apprentissage automatique.

## 2.3 Contraintes supplémentaires

En outre, trois types de contraintes de problème :

- **la contrainte de cardinalité** : qui améliore la prise en considération de la valeur limite du nombre de sommets dans le sous-ensemble après la coupe.
- **la contrainte de connectivité** : qui est exigé pour que les sous-ensembles après la découpe soient encore connectés.
- **la contrainte de capacité** : qui influe le nombre des solutions possibles en limitant la capacité maximale des sommets ou des arêtes.

## 2.4 Synthèse

Ainsi, les problèmes de coupe sont différents selon la structure du graphe, les objectifs poursuivis, et les contraintes appliquées. Cela dit, les différentes approches permettent de moduler la coupe, que ce soit pour optimiser des réseaux, améliorer le clustering ou équilibrer la répartition des sommets.

# Chapitre 3

## Applications et enjeux pratiques

### 3.1 Optimisation des réseaux et gestion des flux

Le problème de coupe est beaucoup utilisé dans l'optimisation des réseaux et la gestion des flux pour améliorer l'efficacité et la sécurité des infrastructures complexes.

#### 3.1.1 Réseaux de télécommunications et informatiques

Dans ces réseaux, le problème de coupe permet :

- De détecter les points critiques et vulnérabilités pour renforcer la robustesse du réseau [wolfram].
- D'optimiser la répartition du trafic pour éviter les goulots d'étranglement et équilibrer la charge [mcours].
- De segmenter un réseau en sous-réseaux pour une gestion plus efficace des ressources [moodle].

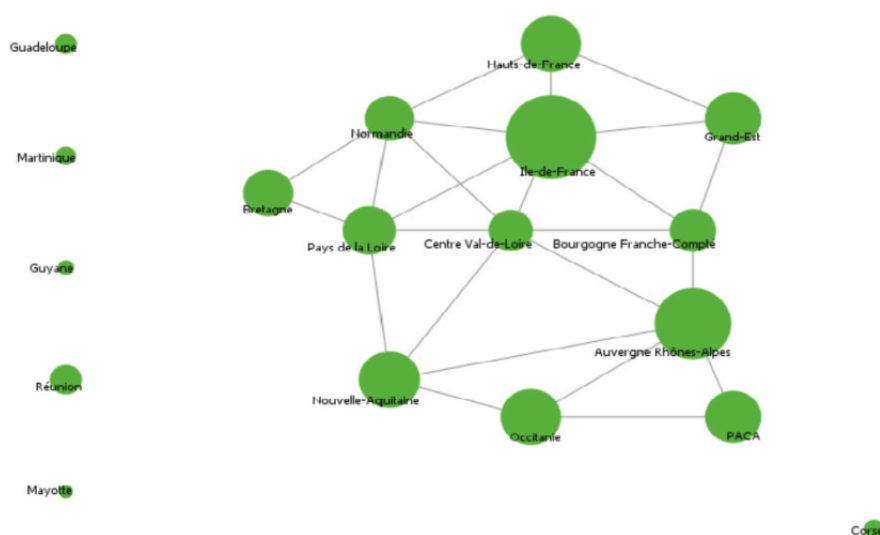


FIGURE 3.1.1 – Representation du reseau des 18 regions francaises graphe pour lequel les noeuds

### 3.1.2 Transport et logistique

Dans la gestion des flux de transport, le problème de coupe est utilisé pour :

- Planifier la distribution des marchandises en minimisant les coûts et en évitant les congestions.
- Identifier les goulets d'étranglement et optimiser la circulation des véhicules [mcours].

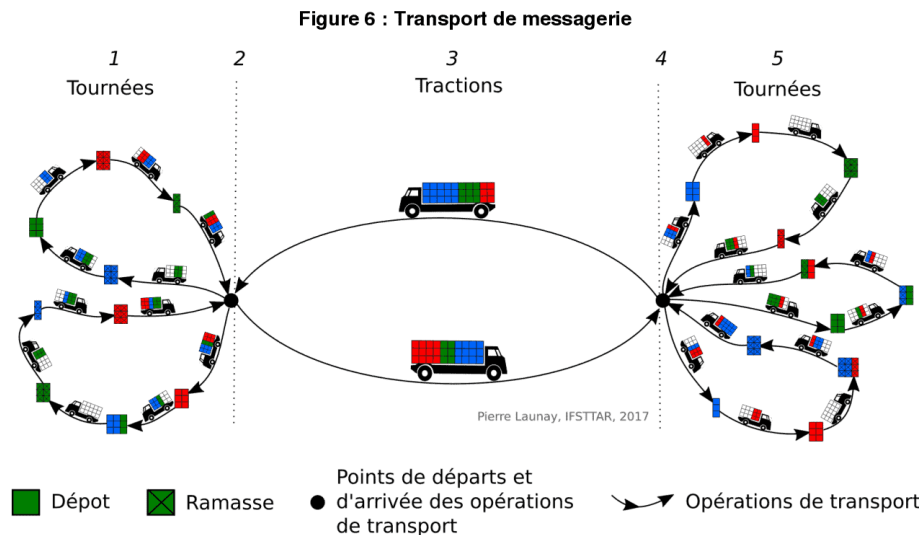


FIGURE 3.1.2 – Le redéploiement technique et organisationnel des réseaux de messagerie dans les territoires. Infrastructures de transport.

## 3.2 Apports en Intelligence Artificielle et Apprentissage Automatique

### 3.2.1 Segmentation d'Images et Clustering

Le problème de coupe joue un rôle essentiel en intelligence artificielle (IA) et en apprentissage automatique, notamment dans :

- **Segmentation d'images** : Il permet de classifier les pixels en régions homogènes afin d'améliorer l'analyse et l'interprétation des images [moodle].
- **Clustering** : Il sert à partitionner un graphe de similarités en groupes cohérents, facilitant ainsi la classification et l'extraction de structures pertinentes [wolfram].

### 3.2.2 Applications en Bioinformatique

En bioinformatique, l'analyse des graphes est une approche clé pour :

- **L'identification de sous-structures importantes** au sein des réseaux de protéines, contribuant à une meilleure compréhension des interactions biologiques.
- **La détection de sous-groupes significatifs** dans les données génétiques, permettant d'identifier des relations et des motifs pertinents pour la recherche biomédicale [mcours].

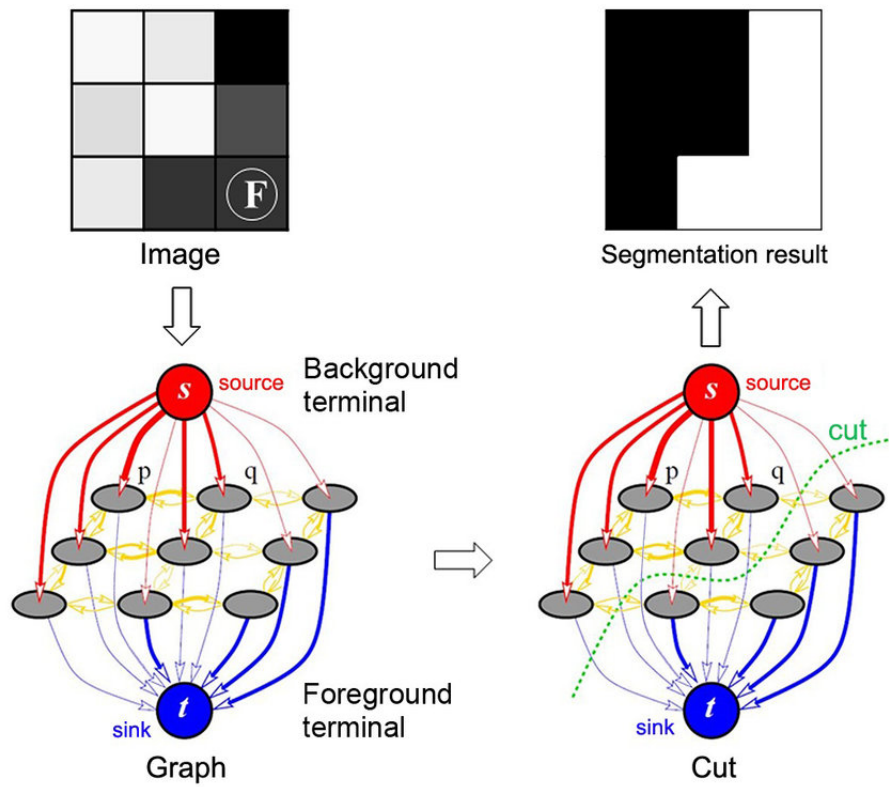


FIGURE 3.2.1 – la segmentation d’images et le clustering dans le cadre du problème du Max-Cut

• Arêtes : interactions entre protéines et gènes

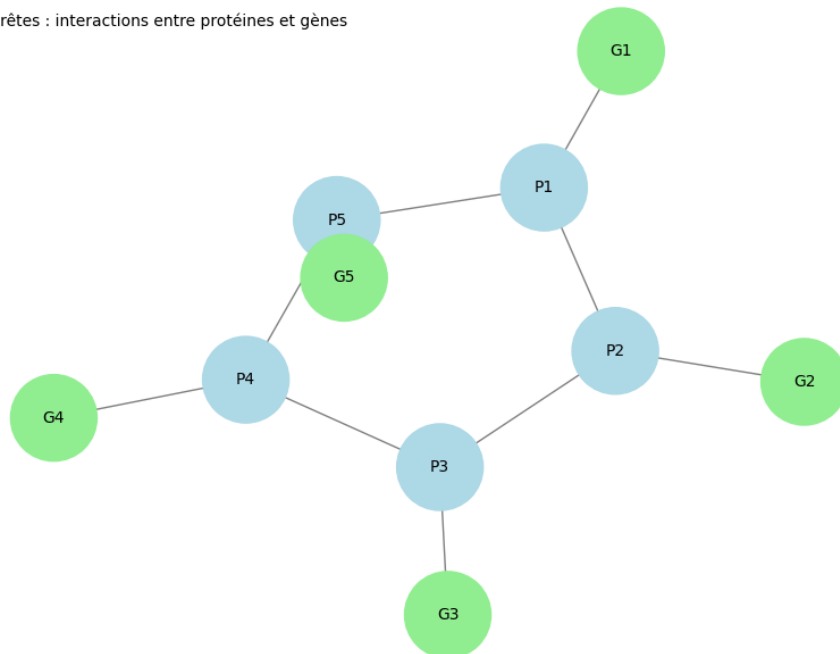
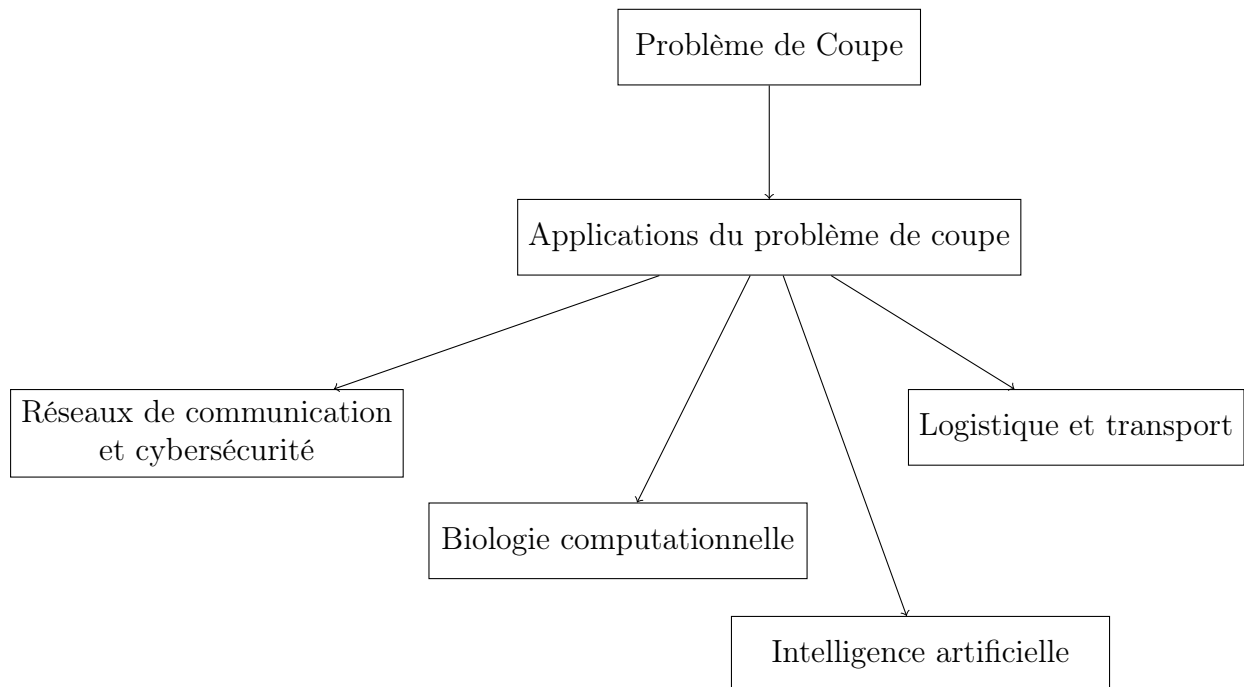


FIGURE 3.2.2 – Analyse des graphes en Bioinformatique

### 3.3 Synthèse des applications



Le problème de coupe est essentiel dans plusieurs domaines :

- **Réseaux de communication et cybersécurité** : Analyse et renforcement de la robustesse.
- **Logistique et transport** : Gestion des flux et optimisation des itinéraires.
- **Intelligence artificielle** : Segmentation et classification des données.
- **Biologie computationnelle** : Identification de structures complexes dans les réseaux biologiques.

Cependant, ces applications rencontrent des défis liés à la complexité computationnelle, aux contraintes réelles et à la scalabilité des méthodes employées.

# Chapitre 4

## Étude de cas et méthodologies de résolution

### 4.1 Présentation de l'étude de cas : Segmentation d'images par Max-Cut

- **Création du graphe** : A partir de l'image, on construit un graphique dans lequel chaque pixel est un noeud et les arêtes sont formées en fonction des voisins de chaque pixel.
- **Attribution des poids** : les poids des arêtes sont calculés en fonction de la similarité entre les pixels. Plus les pixels sont similaires, plus le poids de l'arête est faible.
- **Génération des graines** : Un certain nombre de pixels doivent être marqués par l'utilisateur pour faire partie de l'avant-plan ou de l'arrière-plan. Ces pixels sont reliés respectivement à la source ou au puits par des liens de capacité déterminée.



FIGURE 4.1.1 – Image d'entrée pour la segmentation

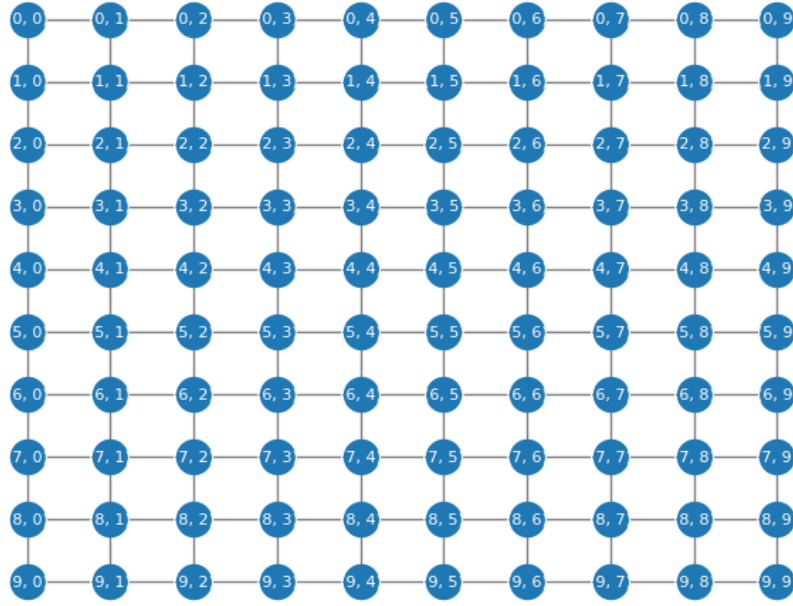


FIGURE 4.1.2 – Construction de graphe à partir de l'image

## 4.2 Exploration des approches exactes, heuristiques et métaheuristiques

### 4.2.1 Approches Exactes

#### Définition de l'approche exacte

Les approches exactes servent à trouver la solution optimale d'un problème en explorant exhaustivement toutes les solutions possibles. Dans le contexte de la segmentation d'images, cela implique l'utilisation de méthodes qui garantissent une séparation optimale des différentes régions de l'image.

---

#### Algorithm 1 Segmentation d'image avec PLNE

---

**Data:** Image  $I$  en niveaux de gris

**Result:** Image segmentée  $S$

Charger l'image  $I$  et obtenir ses dimensions ( $rows, cols$ ) Définir une variable binaire  $x[i, j]$  pour chaque pixel  $(i, j)$  Définir la fonction objectif : Minimiser la somme des différences entre pixels adjacents Définir les contraintes : **foreach** *pixel*  $(i, j)$  **do**

**if**  $I[i, j] > seuil$  **then**

        Fixer  $x[i, j] = 1$

**end**

**end**

Résoudre le problème d'optimisation avec **cvxpy** Obtenir la solution optimale  $x^*$  et reconstruire  $S$  Appliquer un seuillage :  $S[i, j] = 255$  si  $x[i, j] \geq 0.5$ , sinon 0 Enregistrer l'image segmentée  $S$

---

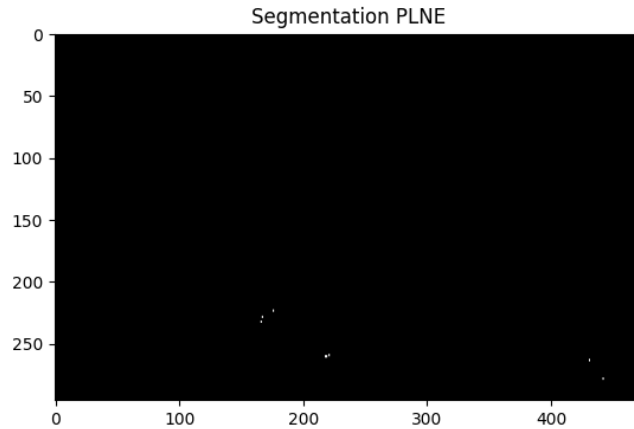


FIGURE 4.2.1 – Application de l’approche exactes pour Max-Cut

## 4.2.2 Approches Heuristiques

### Définition de l’approche Heuristiques

Les méthodes heuristiques sert à trouver une solution acceptable dans un temps raisonnable, mais sans aucune garantie sur l’optimalité. Elles sont souvent basées sur des intuitions ou des règles empiriques et sont particulièrement utiles lorsque les approches exactes sont impraticables en raison de la complexité du problème.

### Algorithme utilisé : Algorithmes gloutons

---

**Algorithm 2** Segmentation d’image avec un algorithme glouton

---

**Data:** Image  $I$  avec  $n$  pixels

**Result:** Ensemble de segments  $S = \{S_1, S_2, \dots, S_k\}$

Charger l’image  $I$  et obtenir ses dimensions ( $rows, cols$ ) Initialiser un ensemble de segments  $S = \{I\}$  (tous les pixels sont dans un seul segment) **while** *condition de convergence non atteinte* **do**

**foreach** *segment*  $S_i \in S$  **do**

        Calculer la similarité entre les pixels de  $S_i$  Choisir un pixel  $p$  à séparer en fonction de la différence de similarité entre les segments Séparer le pixel  $p$  du segment  $S_i$  et créer un nouveau segment  $S_j$  Ajouter le nouveau segment  $S_j$  à l’ensemble des segments  $S$

**end**

**end**

**return** *Ensemble des segments*  $S$

---

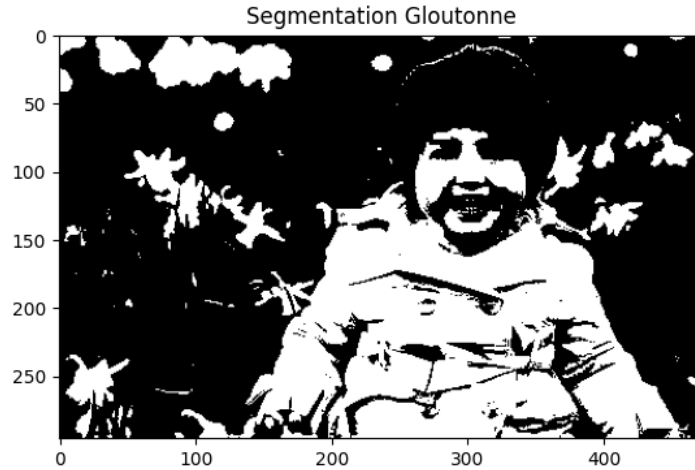


FIGURE 4.2.2 – Application de l’approche heuristique pour Max-Cut

### 4.2.3 Approches Métaheuristiques

#### Définition de l’approche Métaheuristiques

Les approches métaheuristiques sont utilisées pour résoudre des problèmes difficiles comme celui de Max-Cut, offrant des solutions proches de l’optimum.

#### Algorithme utilisé : Algorithme Génétique

---

##### Algorithm 3 Segmentation d’image avec un algorithme génétique

---

**Data:** Image  $I$  avec  $n$  pixels

**Result:** Ensemble de segments  $S = \{S_1, S_2, \dots, S_k\}$

Charger l’image  $I$  et obtenir ses dimensions ( $rows, cols$ ) Initialiser une population de solutions  $P = \{P_1, P_2, \dots, P_m\}$  **while** condition de convergence non atteinte **do**

**foreach** individu  $P_i \in P$  **do**

        Évaluer la qualité de l’individu  $P_i$  en fonction d’une fonction de fitness : Calculer l’homogénéité des segments créés par  $P_i$  Maximiser la différence entre les segments

**end**

    Sélectionner les meilleurs individus selon leur fitness Appliquer les opérateurs génétiques : **foreach** couple d’individus sélectionnés **do**

        Croiser les individus pour créer des enfants Appliquer une mutation sur les enfants avec une probabilité donnée

**end**

    Mettre à jour la population avec les enfants générés

**end**

**return** Ensemble des segments  $S$  correspondant à l’individu optimal

---

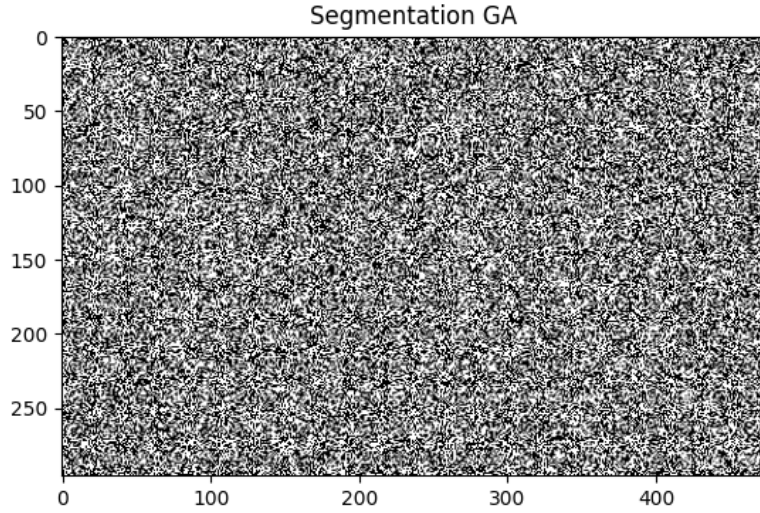


FIGURE 4.2.3 – Application de l’approche Métaheuristiques pour Max-Cut

### 4.3 Résolution numérique des approches de segmentation d’images par Max-Cut

#### 4.3.1 Optimisation par Programmation Linéaire en Nombres Entiers (PLNE)

Considérons une image simple de  $3 \times 3$  pixels :

$$I = \begin{bmatrix} 100 & 150 & 80 \\ 120 & 100 & 140 \\ 90 & 130 & 110 \end{bmatrix}$$

- **Variables binaires** :  $x[i, j] = 1$  si le pixel  $(i, j)$  appartient à l’avant-plan, sinon 0.
- **Formulation du problème** : Objectif : Minimiser les différences entre pixels voisins :

$$\min \sum_{(i,j),(k,l) \in \text{Voisins}} |I[i, j] - I[k, l]| \times |x[i, j] - x[k, l]|$$

- **Contraintes** : Fixer  $x[i, j]$  selon un seuil de segmentation, par exemple 100.

Résolution avec *PuLP* : On obtient la solution  $x$  suivante :

$$x = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Après seuillage :

$$S = \begin{bmatrix} 255 & 0 & 255 \\ 255 & 255 & 0 \\ 255 & 0 & 255 \end{bmatrix}$$

### 4.3.2 Segmentation basée sur des Algorithmes Gloutons

- **Initialisation** : Tous les pixels appartiennent au même segment.
- **Critère de séparation** : Choisir un pixel  $p$  si la différence avec ses voisins dépasse un seuil, par exemple 50.

**Exemple de séparation séquentielle (itération par itération) :**

- *Itération 1* : Séparer les pixels (1, 2) et (2, 3) :

$$S = \begin{bmatrix} 255 & 0 & 255 \\ 255 & 255 & 0 \\ 255 & 255 & 255 \end{bmatrix}$$

- *Itération 2* : Identifier d'autres séparations pour affiner le segment.

L'algorithme converge rapidement avec une segmentation approximative mais rapide.

### 4.3.3 Optimisation avec Algorithmes Génétiques

- **Initialisation de la population** : Créer 10 individus (solutions aléatoires).  
Exemple de population initiale :

$$P_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Fonction de fitness** : Maximiser la similarité des pixels à l'intérieur des segments :

$$\text{Fitness}(P_1) = \sum_{(i,j),(k,l) \in \text{Voisins}} \text{Homogénéité}(i, j, k, l)$$

- **Sélection des meilleurs individus** : Conserver les 5 individus ayant la meilleure fitness.
- **Croisement et mutation** : Croiser entre  $P_1$  et  $P_2$ , puis appliquer une mutation aléatoire avec une probabilité de 0.1.

Après 10 générations, l'algorithme converge vers une solution finale :

$$S = \begin{bmatrix} 255 & 0 & 255 \\ 255 & 255 & 0 \\ 255 & 0 & 255 \end{bmatrix}$$

## 4.4 Synthèse des Résultats

- **PLNE** fournit une solution optimale, mais le temps de calcul augmente rapidement.

- **Algorithmes gloutons** sont rapides mais offrent des résultats approximatifs.
- **Algorithme génétique** atteint un bon compromis en générant des solutions de qualité en un temps raisonnable.

# Chapitre 5

## Analyse critique et perspectives

### 5.1 Comparaison des méthodes

Dans cette section, nous analysons trois méthodes de segmentation : PLNE, algorithme glouton et algorithme génétique. Les tableaux suivants résument les avantages, inconvénients et pistes d'amélioration pour chaque méthode.

| Critères                 | PLNE (Programmation Linéaire en Nombres Entiers)      |
|--------------------------|-------------------------------------------------------|
| Précision                | Très élevée (optimale)                                |
| Temps de calcul          | Très lent pour grandes images                         |
| Complexité algorithmique | Élevée                                                |
| Robustesse               | Stable, mais difficile à généraliser                  |
| Interprétabilité         | Facile à expliquer mais computationnellement complexe |

TABLE 5.1 – Analyse de la méthode PLNE

| Critères                 | Algorithme Glouton                    |
|--------------------------|---------------------------------------|
| Précision                | Moyenne                               |
| Temps de calcul          | Rapide                                |
| Complexité algorithmique | Faible                                |
| Robustesse               | Peut échouer sur des images complexes |
| Interprétabilité         | Simple à comprendre                   |

TABLE 5.2 – Analyse de l'algorithme glouton

| Critères                 | Algorithme Génétique                             |
|--------------------------|--------------------------------------------------|
| Précision                | Variable (dépend des paramètres)                 |
| Temps de calcul          | Moyen à lent                                     |
| Complexité algorithmique | Moyenne à élevée                                 |
| Robustesse               | Dépend de la diversité de la population initiale |
| Interprétabilité         | Difficile à interpréter                          |

TABLE 5.3 – Analyse de l’algorithme génétique

## 5.2 Conclusion et perspectives

- La méthode **PLNE** accorde des résultats optimaux mais n’est pas adaptée aux images de grandes taille en raison de sa complexité computationnelle.
- L’**algorithme glouton** est une approche rapide et simple, mais manque de précision et dépend du choix initial des centres.
- L’**algorithme génétique** permet une flexibilité importante mais nécessite un bon réglage des hyperparamètres pour obtenir des résultats satisfaisants.

### Pistes d’amélioration :

- Améliorer l’efficacité du PLNE en intégrant des heuristiques.
- Introduire une approche hybride combinant la rapidité de l’algorithme glouton et l’exploration de l’algorithme génétique.
- Optimiser l’algorithme génétique en adaptant dynamiquement ses paramètres pour améliorer la convergence.

# Chapitre 6

## Implémentation Python

### 6.1 Bibliothèques Utilisées

Les bibliothèques utilisées pour l'implémentation sont les suivantes :

- **OpenCV (cv2)** : Utilisée pour lire et manipuler des images.
- **NumPy** : Fournit des outils efficaces pour la manipulation des tableaux de pixels.
- **Matplotlib** : Utilisée pour l'affichage des résultats de segmentation.
- **PuLP** : Bibliothèque pour la résolution de problèmes d'optimisation linéaire en nombres entiers.
- **DEAP (Distributed Evolutionary Algorithms in Python)** : Fournit des outils pour l'implémentation d'algorithmes génétiques.
- **Random** : Génère des valeurs aléatoires utilisées dans l'initialisation des populations et des segments.

### 6.2 Chargement de l'Image

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Charger l'image
6 image_path = r'C:\Users\HP\Desktop\M1\S1\optimisation
7             combinatoire\image.jpg'
8 image = cv2.imread(image_path)
9 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
10 gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

### 6.3 Segmentation par Programmation Linéaire en Nombres Entiers (PLNE)

```
1 from pulp import LpMaximize, LpProblem, LpVariable
```

```

2
3 def integer_programming_segmentation(image_data):
4     rows, cols = image_data.shape
5     model = LpProblem(name="image_segmentation", sense=LpMaximize)
6     variables = [[LpVariable(f"x_{i}_{j}", cat="Binary") for j in
7         range(cols)] for i in range(rows)]
8
9     # Exemple de contrainte simple : maximiser le contraste entre
10    pixels voisins
11    objective = sum(abs(int(image_data[i, j]) -
12        int(image_data[i+1, j])) * variables[i][j]
13        for i in range(rows-1) for j in range(cols-1))
14    model += objective
15    model.solve()
16
17    result = np.array([[1 if variables[i][j].varValue else 0 for
18        j in range(cols)] for i in range(rows)])
19    return result
20
21 def run_integer_programming(image_data):
22     segmentation_plne =
23         integer_programming_segmentation(image_data)
24     plt.imshow(segmentation_plne, cmap='gray')
25     plt.title("Segmentation PLNE")
26     plt.show()
27
28 # Execution
29 run_integer_programming(gray)

```

## 6.4 Segmentation par Algorithme Glouton

```

1 def greedy_segmentation(image_data, n_segments=2):
2     rows, cols = image_data.shape
3     centers = np.random.choice(np.arange(image_data.min(),
4         image_data.max()), size=n_segments, replace=False)
5     segmentation = np.zeros((rows, cols), dtype=int)
6
7     for _ in range(100):
8         for i in range(rows):
9             for j in range(cols):
10                pixel_value = image_data[i, j]
11                distances = np.abs(centers - pixel_value)
12                segmentation[i, j] = np.argmin(distances)
13
14    # Recalculer les centres comme la moyenne des pixels
15    assign s
16    for k in range(n_segments):

```

```

15         segment_pixels = image_data[segmentation == k]
16         if len(segment_pixels) > 0:
17             centers[k] = np.mean(segment_pixels)
18
19     return segmentation
20
21 def run_greedy_segmentation(image_data):
22     segmentation_glouton = greedy_segmentation(image_data)
23     plt.imshow(segmentation_glouton, cmap='gray')
24     plt.title("Segmentation Gloutonne")
25     plt.show()
26
27 # Execution
28 run_greedy_segmentation(gray)

```

## 6.5 Segmentation par Algorithme Génétique

```

1 from deap import base, creator, tools, algorithms
2 import random
3
4 def genetic_algorithm_segmentation(image_data, n_clusters=2,
5     n_gen=10, pop_size=20):
6     rows, cols = image_data.shape
7
8     # Initialiser la structure de l'algorithme genetique
9     creator.create("FitnessMax", base.Fitness, weights=(1.0,))
10    creator.create("Individual", list, fitness=creator.FitnessMax)
11    toolbox = base.Toolbox()
12    toolbox.register("attr_bool", random.randint, 0, 1)
13    toolbox.register("individual", tools.initRepeat,
14        creator.Individual, toolbox.attr_bool, n=rows*cols)
15    toolbox.register("population", tools.initRepeat, list,
16        toolbox.individual)
17
18    def eval_segmentation(individual):
19        mask = np.array(individual).reshape(rows, cols)
20        contrast = np.sum(np.abs(image_data[:-1, :] -
21            image_data[1:, :]) * mask[:-1, :])
22        return contrast,
23
24    toolbox.register("evaluate", eval_segmentation)
25    toolbox.register("mate", tools.cxTwoPoint)
26    toolbox.register("mutate", tools.mutFlipBit, indpb=0.1)
27    toolbox.register("select", tools.selTournament, tournsize=3)
28
29    # Ex cuter l'volution
30    population = toolbox.population(n=pop_size)

```

```

27     algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2,
28         ngen=n_gen, verbose=False)
29     best_ind = tools.selBest(population, 1)[0]
30     segmentation = np.array(best_ind).reshape(rows, cols)
31     return segmentation
32
33 def run_genetic_algorithm(image_data):
34     segmentation_ga = genetic_algorithm_segmentation(image_data)
35     plt.imshow(segmentation_ga, cmap='gray')
36     plt.title("Segmentation GA")
37     plt.show()
38
39 # Execution
run_genetic_algorithm(gray)

```

# Conclusion

Au cours de ce travail de recherche, nous avons étudié de manière exhaustive la problématique de la coupe maximale et de ses différentes applications, parmi lesquelles nous nous intéressons notamment à la segmentation d'images. Nous avons eu l'occasion de rappeler ses fondements mathématiques, ses variantes, ainsi que les différentes façons de l'aborder via des méthodes exactes, des heuristiques ou encore des métaheuristiques.

L'analyse des résultats montre que que les approches a ses avantages et ses désavantages : les méthodes exactes garantissent des solutions optimales mais coûteuses en termes de calcul, tandis que les heuristiques et métaheuristiques offrent des alternatives plus rapides et adaptées aux grands ensembles de données. En particulier, l'algorithme génétique a démontré un bon équilibre entre précision et temps de calcul, ce qui le rend particulièrement intéressant pour les applications pratiques.

Cependant, d'autres voies sont encore à explorer. L'utilisation de méthodes hybrides combinant la rapidité de traitement heuristique et la précision de la méthode exacte pourrait performancesles performances. Par ailleurs, l'optimisation des paramètres des métaheuristiques pourrait obtenir des solutions plus robustes.

Enfin, au-delà de la segmentation d'images, ces techniques d'optimisation combinatoire peuvent être appliquées à d'autres domaines comme la bio-informatique, la cybersécurité ou encore la gestion des réseaux, ouvrant ainsi la voie à de nouvelles recherches et applications innovantes.

# Bibliographie

- [1] [Auteur inconnu, \*Titre du document\*, disponible en ligne](#)
- [2] [Wikipédia, \*Coupe maximum\*, disponible en ligne](#)
- [3] [Wikipédia, \*Graphe biparti\*, disponible en ligne](#)
- [4] [Wolfram - Max Cut Problem](#)
- [5] [Cours en optimisation - Mcours.net](#)
- [6] [Leçon 6 - Université Numérique](#)
- [7] [Représentation du réseau des 18 régions françaises graphe pour lequel les noeuds](#)
- [8] [Pierre Launay. Le redéploiement technique et organisationnel des réseaux de messagerie dans les territoires. Infrastructures de transport. Université Paris-Est, 2018.](#)
- [9] [Wikipédia, \*graphe planaire\*, disponible en ligne](#)
- [10] [la segmentation d'images et le clustering dans le cadre du problème du Max-Cut](#)
- [11] [définition de la segmentation d'images](#)
- [12] [Segmentation d'images par Max-Cut](#)